8

Функции

Добавянето на нови оператори и функции в приложенията, реализирани на езика C++, се осъществява чрез функциите. Те са основни структурни единици, от които се изграждат програмите на езика. Всяка функция се състои от множество от оператори, оформени подходящо за да се използват като обобщено действие или операция. След като една функция бъде дефинирана, тя може да бъде изпълнявана многократно за различни входни данни.

Програмите на езика C++ се състоят от една или повече функции. Сред тях задължително трябва да има точно една с име main и наречена **главна функция**. Тя е първата функция, която се изпълнява при стартиране на програмата. Главната функция от своя страна може да се обръща към други функции. Нормалното изпълнение на програмата завършва с изпълнението на главната функция (Възможно е изпълнението да завърши принудително с изпълнението на функция, различна от главната).

Използването на функции има следните предимства:

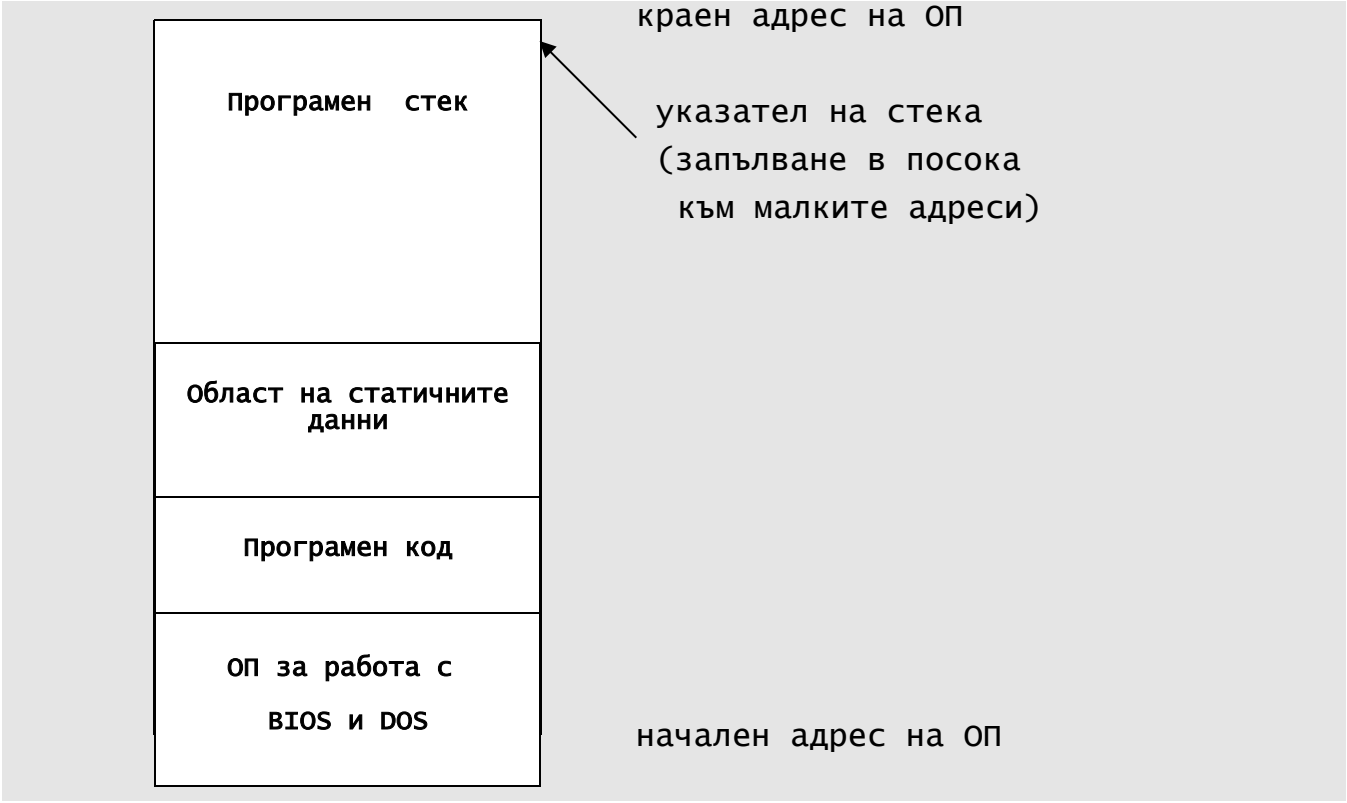
- Програмите стават ясни и лесни за тестване и модифициране.
- Избягва се многократното повтаряне на едни и същи програмни фрагменти. Те се дефинират еднократно като функции, след което могат да бъдат изпълнявани произволен брой пъти.
- Постига се икономия на памет, тъй като кодът на функцията се съхранява само на едно място в паметта, независимо от броя на нейните изпълнения.

Ще разгледаме най-общо разпределението на оперативната памет за изпълнима програма на C++. Чрез няколко примерни програми ще покажем

дефинирането, обръщението и изпълнението на функции, след което ще направим съответните обобщения.

8.1 Разпределение на ОП за изпълнима програма

Разпределението на ОП зависи от изчислителната система, от типа на операционната система, а също от модела памет. Най-общо се състои от: *програмен код, област на статичните данни, област на динамичните данни* и *програмен стек* (Фиг. 8.1).



Фиг. 8.1 Разпределение на ОП

Програмен код

В тази част е записан изпълнимият код на всички функции, изграждащи потребителската програма.

Област на статичните данни

В нея са записани глобалните обекти (в широкия смисъл на думата) на програмата.

Област на динамичните данни

За реализиране на динамични структури от данни (списъци, дървета, графи, ...) се използват средства за динамично разпределение на паметта. Чрез тях се заделя и освобождава памет в процеса на изпълнение на програмата, а не преди това (при компилирането ѝ). Тази памет е от областта на динамичните данни.

Програмен стек

Този вид памет съхранява данните на функциите на програмата. Стекът е динамична структура, организирана по правилото "последен влязъл – пръв излязъл". Той е редица от елементи с пряк достъп до елементите от единия си край, наречен **връх**. Достъпът се реализира чрез указател. Операцията включване се осъществява само пред елемента от върха, а операцията изключване – само за елемента от върха.

Елементите на програмния стек са "блокове" от памет, съхраняващи данни, дефинирани в някаква функция. Наричат се **стекови рамки**.

8.2 Примери за програми, които дефинират и използват функции

Задача 68. Да се напише програма, която въвежда стойности на естествените числа a, b, c и d и намира и извежда най-големият общ делител на числата a и b, след това на c и d и накрая на a, b, c и d.

Програма Zad68.cpp решава задачата. Тя се състои от две функции: gcd и main. Функцията gcd(x, y) намира най-големия общ делител на естествените числа x и y. Тъй като main се обръща към (извиква) gcd, функцията gcd трябва да бъде известна преди функцията main. Най-лесният начин да се постигне това е във файла, съдържащ програмата, първо да се постави дефиницията на gcd, а след това тази на main. Ще бъде показан алтернативен начин по-късно.

Описанието на функцията gcd прилича на това на функцията main. Състои се от заглавие

```
int gcd(int x, int y)
```

и тяло

```
{while (x != y)
 if (x > y) x = x-y; else y = y-x;
```

```
 return x;
 }
```

Заглавието определя, че gcd е име на двуаргументна целочислена функция с цели аргументи, т.е.

gcd: int x int ⟶ int

Името е произволен идентификатор. В случая е направен мнемонически избор. Запазената дума int пред името на функцията е типа ѝ (по-точно е типа на резултата на функцията). В кръгли скоби и отделени със запетая са описани параметрите x и y на gcd. Те са различни идентификатори. Предшестват се от типовете си. Наричат се **формални параметри за функцията**.

Тялото на функцията е блок, реализиращ алгоритъма на Евклид за намиране на най-големия общ делител на естествените числа x и y. Завършва с оператора

return x;

чрез който се прекратява изпълнението на функцията като стойността на израза след return се връща като стойност на gcd в мястото, в случая в main, в което е направено обръщението към нея.

```
// Program Zad68.cpp
#include <iostream.h>
int gcd(int x, int y)
{while (x != y)
 if (x > y) x = x-y; else y = y-x;
 return x;
 }
 int main()
 {cout << "a, b, c, d= ";
  int a, b, c, d;
  cin >> a >> b >> c >> d;
  if (!cin || a < 1 || b < 1 || c < 1 || d < 1)
  {cout << "Error! \n";
   return 1;
  }
  int r = gcd(a, b);
  cout << "gcd{" << a << ", " << b << "}= " << r << "\n";
```

```
  int s = gcd(c, d);
  cout << "gcd{" << c << ", " << d << "}= " << s << "\n";
  cout << "gcd{" << a << ", " << b << ", " << c << ", "
       << d << "}= " << gcd(r, s) << "\n";
   return 0;
 }
```

Изпълнение на програма Zad68.cpp

Дефинициите на функциите main и gcd се записват в областта на паметта, определена за програмния код. Изпълнението на програмата започва с изпълнение на функцията main. Фрагментът

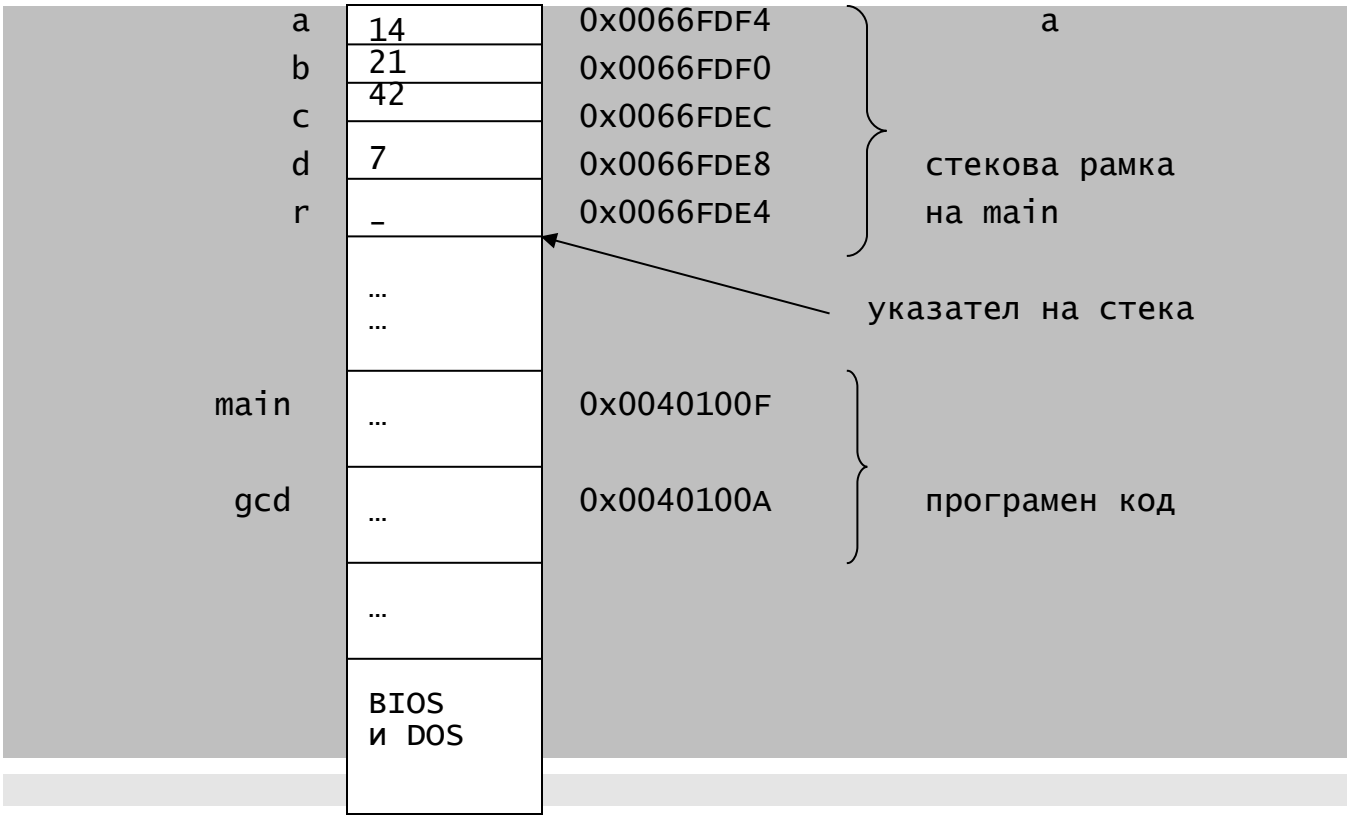
```
cout << "a, b, c, d= ";
int a, b, c, d;
cin >> a >> b >> c >> d;
if (!cin || a < 1 || b < 1 || c < 1 || d < 1)
{cout << "Error \n";
 return 1;
}
```

дефинира и въвежда стойности на целите променливи a, b, c и d като осигурява да са естествени числа. Нека за a, b, c и d са въведени 14, 21, 42 и 7 съответно. В тази последователност те се записват в дъното на програмния стек (Фиг. 8.2). Така на дъното на стека се оформя "блок" от памет за main с достатъчно големи размери, който освен променливите от main съдържа и някои "вътрешни" данни. Този блок се нарича **стекова рамка на main**.

Операторът

```
int r = gcd(a, b);
```

дефинира цялата променлива r като в стековата рамка на main, веднага след променливата d отделя 4в, в които ще запише резултатът от обръщението gcd(a, b) към функцията gcd. Променливите a и b се наричат **фактически параметри за това обръщение**. Забелязваме, че типът им е същия като на съответните им формални параметри x и y.



Фиг. 8.2 Разпределение на ОП програмата Zad68.cpp

Обръщение към gcd(a, b)

В програмния стек се генерира нов блок памет – стекова рамка за функцията gcd. В него се записват формалните и локалните параметри на gcd, а също и някои "вътрешни" данни като return-адреса и адреса на стековата рамка на main. Указателят на стека се премества след тази стекова рамка.

Обръщението се осъществява по следния начин:

а) Свързване на формалните с фактическите параметри

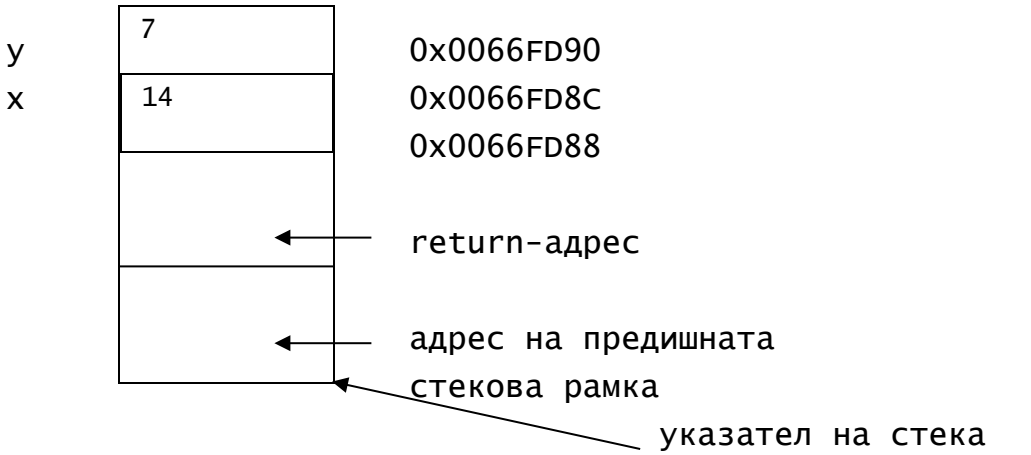
В стековата рамка на gcd, се отделят по 4 байта за формалните параметри x и y в обратен ред на реда, в които са записани в заглавието. В тази памет се **откопират стойностите** на съответните им фактически параметри. Отделят се също 4B за т. нар. return-адрес, адреса на мястото в main, където ще се върне резултатът, а също се отделя памет, в която се записва адресът на предишната стекова рамка, т.е.

памет за gcd (I-во обръщение към него)

б) Изпълнение на тялото на gcd

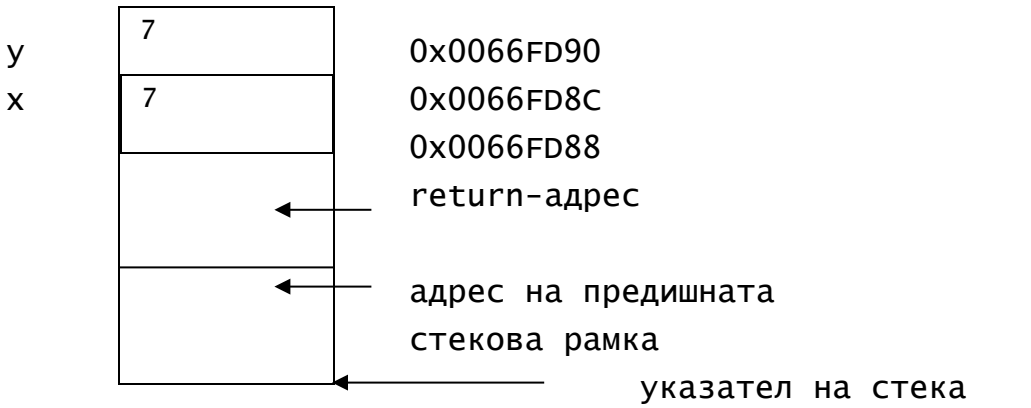
Тъй като е в сила y > x, стойността на y се променя на 7, т.е.

памет в стека за gcd



Сега пък е в сила x > y, което води до промяна на стойността на x на 7, т.е.

памет в стека за gcd



Операторът за цикъл завършва изпълнението си. Изпълнението на оператора

```
return x;
```

преустановява изпълнението на gcd като връща в main в мястото на прекъсването (return-адреса) стойността 7 на обръщението gcd(a, b). Отделената за gcd стекова рамка се освобождава. Указателят на стека се установява в края на стековата рамка на main. Изпълнението на програмата продължава с инициализацията на r. Резултатът от обръщението gcd(14, 21) се записва в отделената за r памет.

Операторът

```
cout << "gcd{" << a << ", " << b << "}= " << r << "\n";
```

извежда получения резултат.

Изпълнението на останалите обръщения към gcd се реализира по същия начин. При обръщението към всяко от тях в стека се създава стекова рамка на gcd, а след завършване на обръщението, рамката се освобождава. При достигане до оператора return 0; от main, се освобождава и стековата рамка на main.

Функцията gcd реализира най-простото и "чисто" дефиниране и използване на функции – получава входните си стойности единствено чрез формалните си параметри и връща резултата от работата си чрез оператора return. Забелязваме, че обръщението gcd(a, b) работи с копия на стойностите на a и b, запомнени в x и y, а не със самите a и b. В процеса на изпълнение на тялото на gcd, стойностите на x и y се променят, но това не оказва влияние на стойностите на фактическите параметри a и b.

Такова свързване на формалните с фактическите параметри се нарича **свързване по стойност** или още **предаване на параметрите по стойност**. При него фактическите параметри могат да бъдат не само променливи, но и изрази от типове, съвместими с типовете на съответните формални параметри. Обръщението gcd(gcd(a, b), gcd(c, d)) е коректно и намира най-големия общ делител на a, b, c и d.

В редица случаи се налага функцията да получи входа си чрез някои от формалните си параметри и да върне резултат не по обичайния начин – чрез оператора return, а чрез същите или други параметри. Задача 69 дава пример за това.

Задача 69. Да се напише програма, която въвежда стойности на реалните променливи a, b, c и d, след което разменя стойностите на a и b и на c и d съответно.

Ако дефинираме функция swapi(double* x, double* y), която разменя стойностите на реалните променливи, към които сочат указателите x и y, обръщението swapi(&a, &b) ще размени стойностите на a и b, а обръщението swapi(&c, &d) ще размени стойностите на c и d. Програма Zad69.cpp решава задачата. Тя се състои от функциите: swapi и main. Тъй като main се обръща към (извиква) swapi, функцията swapi трябва да бъде известна преди функцията main. Затова във файла, съдържащ програмата, първо е поставена функцията swapi, а след това - main.

```
// Program Zad69.cpp
#include <iostream.h>
#include <iomanip.h>
void swapi(double* x, double* y)
{double work = *x;
 *x = *y;
 *y = work;
 return;
}
int main()
{cout << "a, b, c, d= ";
 double a, b, c, d;
 cin >> a >> b >> c >> d;
 cout << setprecision(2) << setiosflags(ios :: fixed);
 cout << setw(10) << a << setw(10) << b
      << setw(10) << c << setw(10) << d << '\n';
 swapi(&a, &b);
 swapi(&c, &d);
 cout << setw(10) << a << setw(10) << b
      << setw(10) << c << setw(10) << d << '\n';
 return 0;
}
```

Функцията swapi има подобна структура като на gcd. Но и заглавието, и тялото й са по-различни. Типът на swapi е указан чрез запазената дума void. Това означава, че функцията не връща стойност чрез оператора return. Затова в тялото на swapi е пропуснат изразът след return (възможно е да бъде пропуснат и самия return). Формалните параметри x и y са указатели към типа double, а в тялото се работи със съдържанията на указателите.

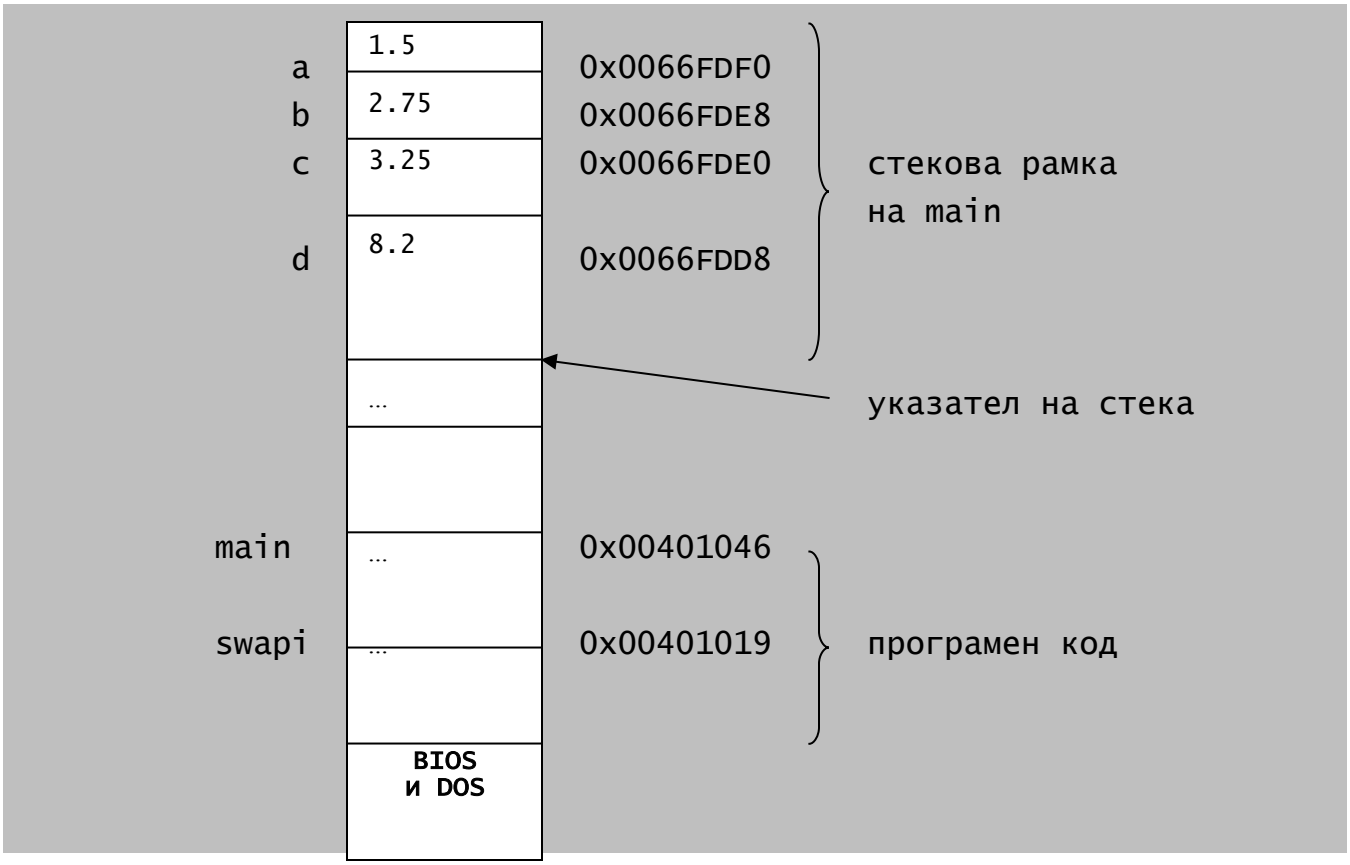
Забелязваме също, че обръщенията към swapi в main

```
swapi(&a, &b);
swapi(&c, &d);
```

не участват като аргументи на операции, а са оператори.

Изпълнение на програма Zad69.cpp

Дефинициите на функциите main и swapi се записват в областта на паметта, определена за програмния код. Изпълнението на програмата започва с изпълнение на функцията main.



Фиг. 8.3 Разпределение на паметта за swapi

Фрагментът

```
cout << "a, b, c, d= ";
double a, b, c, d;
cin >> a >> b  >> c >> d;
cout << setprecision(2) << setiosflags(ios :: fixed);
cout << setw(10) << a << setw(10) << b
     << setw(10) << c << setw(10) << d << '\n';
```

дефинира и въвежда стойности за реалните променливи a, b, c и d и ги извежда върху екрана според дефинираното форматиране. Нека за стойности на a, b, c и d са въведени 1.5, 2.75, 3.25 и 8.2 съответно (Фиг. 8.3).

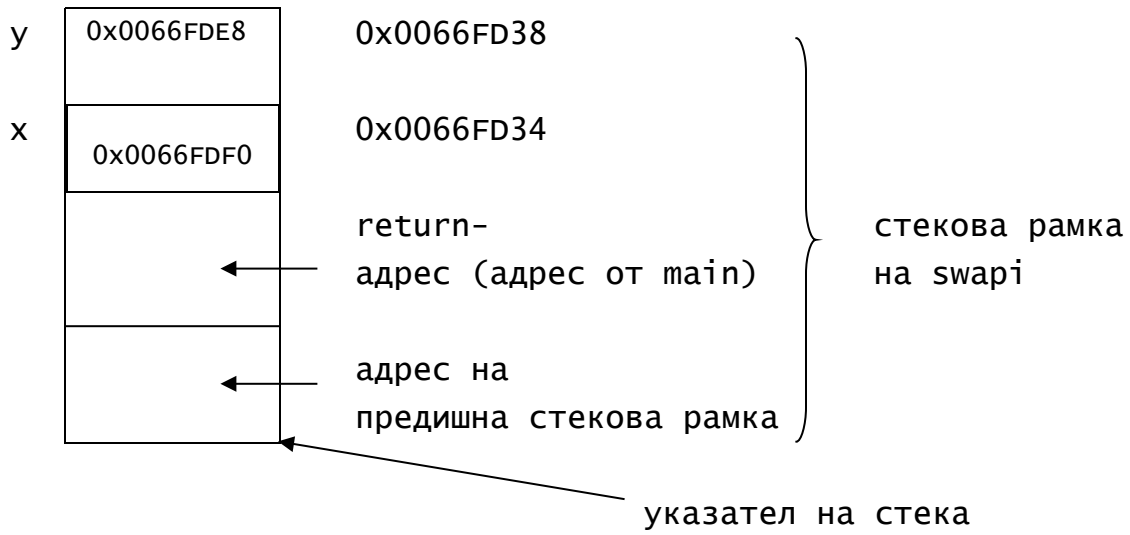
Обръщението

```
swapi(&a, &b);
```

се изпълнява по следния начин:

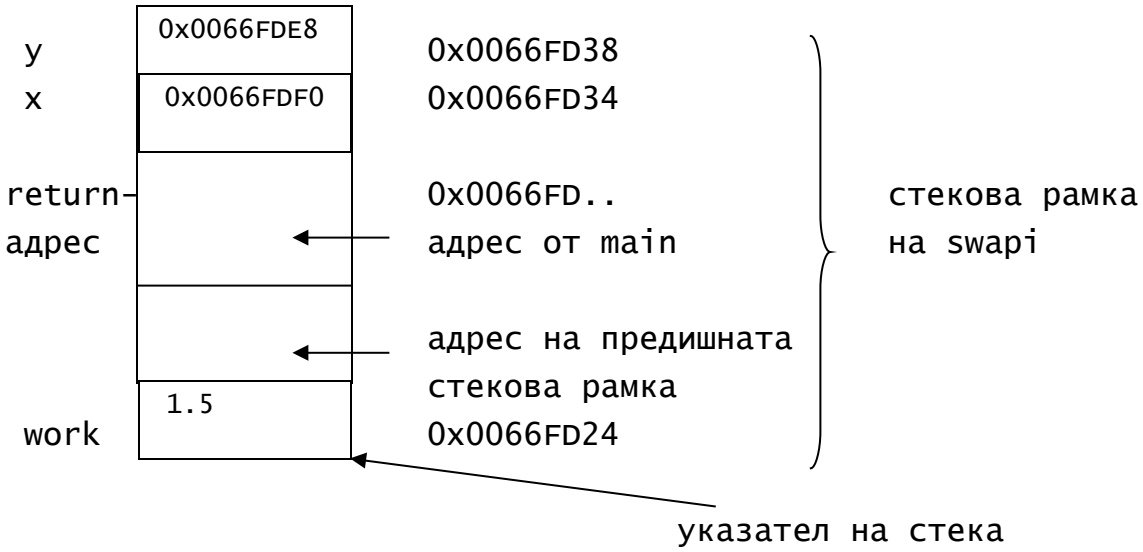
а) Свързване на формалните с фактическите параметри

В стека се конструира нова рамка – рамката на swapi. Указателят на стека се установява след нея. Отделят се по 4 байта за формалните параметри x и y, в която памет се **записват адресите** на съответните им фактически параметри, още 4В, в които се записва адресът на swapi(c, d), от където трябва да се продължи изпълнението на main (return-адреса), а също и памет, в която се записва адресът на предишната стекова рамка (в случая на main).



б) Изпълнение на тялото на swapi

Изпълнява се като блок. За реалната променлива work се отделят 8 байта в стековата рамка на swapi, в които се записва съдържанието на x, в случая 1.5, т.е.



Операторът

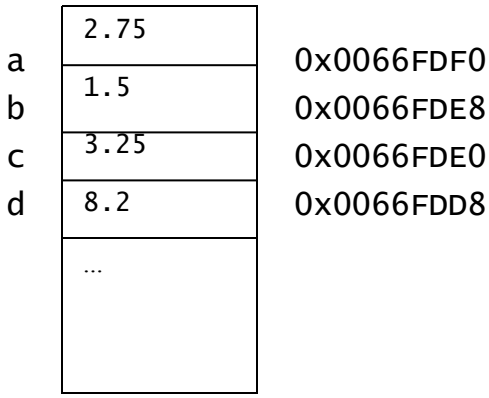
```
*x = *y;
```

променя съдържанието на x с това на y, а операторът

```
*y = work;
```

променя съдържанието на y като го свързва със стойността на work, т.е.

стекова рамка на main



Операторът return; прекъсва работа на на swapi и предава управлението в точката на извикването му в главната функция (return-адреса). Стековата рамка, отделена за swapi се освобождава. Указателят на

стека сочи стековата рамка на main. В резултат стойностите на променливите a и b са разменени.

Обръщението swapi(c, d) се изпълнява по аналогичен начин. За нея се генерира нова стекова рамка (на същите адреси), която се освобождава когато изпълнението на swapi завърши.

Функцията swapi получава входните си стойности чрез формалните си параметри и връща резултата си чрез тях. Забелязваме, че обръщението swapi(&a, &b) работи не с копия на стойностите на a и b, а с адресите им. В процеса на изпълнение на тялото се променят стойностите на фактическите параметри a и b при първото обръщение към нея и на c и d – при второто.

Такова свързване на формалните с фактическите параметри се нарича **свързване на параметрите по указател** или още **предаване на параметрите по указател** или **свързване по адрес**. При този вид предаване на параметрите, фактическите параметри задължително са променливи или адреси на променливи.

Освен тези два начина на предаване на параметри, в езика C++ има още един – **предаване на параметри по псевдоним**. Той е сравнително по-удобен от предаването по указател и се предпочита от програмистите.

Ще го илюстрираме чрез същата задача. Програма Zad69_1.cpp реализира функция swapi, в която предаването на параметрите е по псевдоним.

```
// Program Zad69_1.cpp
#include <iostream.h>
#include <iomanip.h>
void swapi(double& x, double& y)
{double work = x;
 x = y;
 y = work;
 return;
}
int main()
{cout << "a, b, c, d= ";
 double a, b, c, d;
 cin >> a >> b >> c >> d;
```

```
    cout << setprecision(2) << setiosflags(ios :: fixed);
    cout << setw(10) << a << setw(10) << b
         << setw(10) << c << setw(10) << d << '\n';
    swapi(a, b);
    swapi(c, d);
    cout << setw(10) << a << setw(10) << b
         << setw(10) << c << setw(10) << d << '\n';
    return 0;
   }
```

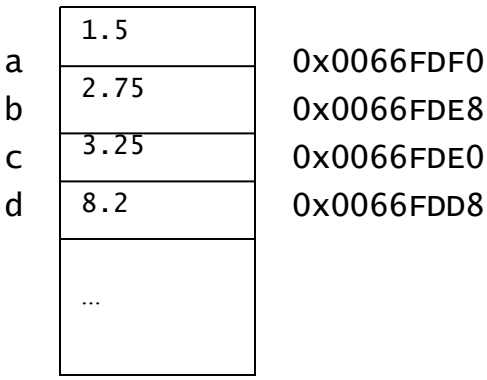
Ще проследим изпълнението и на тази модификация.

Изпълнението на програмата започва с изпълнение на функцията main. Фрагментът

```
   cout << "a, b, c, d= ";
   double a, b, c, d;
   cin >> a >> b >> c >> d;
   cout << setprecision(2) << setiosflags(ios :: fixed);
   cout << setw(10) << a << setw(10) << b
        << setw(10) << c << setw(10) << d << '\n';
```

дефинира и въвежда стойности за реалните променливи a, b, c и d и ги извежда върху екрана според дефинираното форматиране. Нека за стойности на a, b, c и d отново са въведени 1.5, 2.75, 3.25 и 8.2 съответно. След обработката му в стека се конструира стековата рамка на main.

памет на main



Обръщението

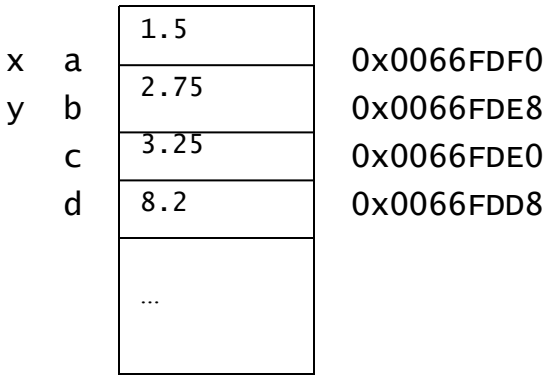
```
   swapi(a, b);
```

се изпълнява по следния начин:

а) Свързване на формалните с фактическите параметри

За целта се генерира нова стекова рамка – рамката на swapi. Указателят на стека сочи тази рамка. Тъй като формалните параметри x и y са псевдоними на променливите a и b съответно, за тях памет в стековата рамка на swapi не се отделя. Параметърът x "прелита" и се "закачва" за фактическия параметър a и аналогично y "прелита" и се "закачва" за фактическия параметър b от стековата рамка на main. Така всички действия с x и y в swapi се изпълняват с фактическите параметри a и b от main съответно.

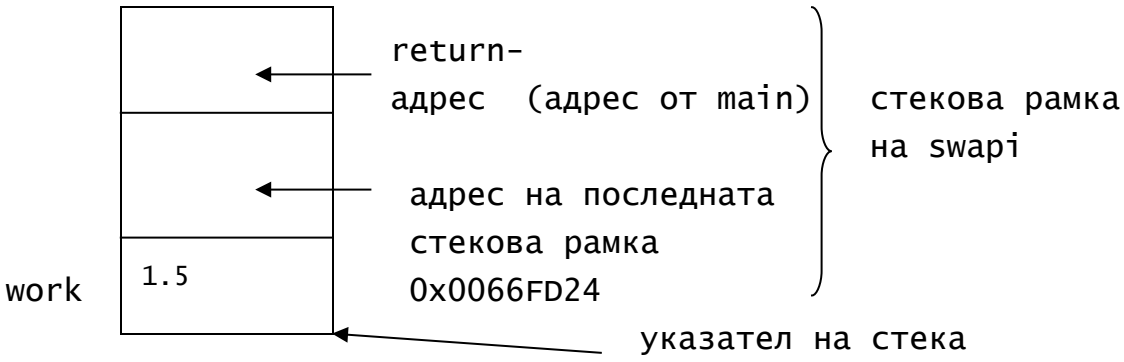
памет на main



б) Изпълнение на тялото на swapi

Изпълнява се като блок. В рамката на swapi, за реалната променлива work се отделят 8 байта, в които се записва стойността на x, в случая 1.5, т.е.

стекова рамка на swapi



Операторът

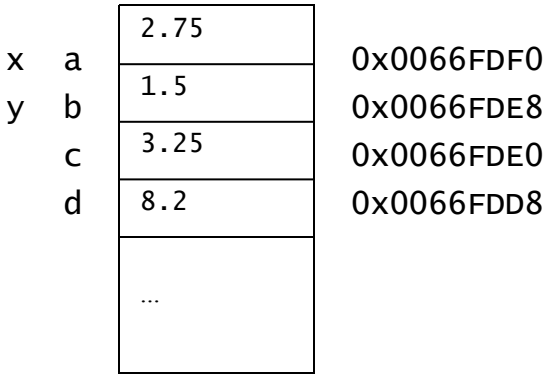
x = y;

присвоява на a стойността на b, а операторът

```
y = work;
```

променя стойността на променливата b като й присвоява стойността на work, т.е.

стекова рамка на main



Операторът return; прекъсва работа на на swapi и предава управлението на return-адреса от главната функция. Стековата рамка на swapi се освобождава. Указателят на стека сочи стековата рамка на main. В резултат, стойностите на променливите a и b са разменени. Променливите a и b са "освободени от" x и y. Следва изпълнение на обръщението

```
swapi(c, d);
```

което се реализира по същия начин (даже стековата му рамка се разполага на същите адреси в стека).

Забелязваме, че фактическите параметри, съответстващи на формални параметри-псевдоними са променливи.

Тази реализация на swapi е по-ясна и удобна от съответната й с указатели. Тялото й реализира размяна на стойностите на две реални променливи без да се налага използването на адреси.

Нека в тялото на main на Zad69_1.cpp преди оператора return; включим фрагмента:

```
int m, n;
cin >> m >> n;
swapi(m, n);
cout << setw(10) << m << setw(10) << n << "\n";
```

Ще отбележим, че m и n са от тип int, а формалните параметри на swapi са псевдоними на тип double. Някои реализации (в това число Visual C+

+ 6.0) ще сигнализират грешка на третата линия – невъзможност за преобразуване на параметър от int в double &, други обаче ще имат нормално поведение, но няма да разменят стойностите на m и n. Последното е така, тъй като при несъответствие на типа на псевдонима с типа на инициализатора, в стековата рамка на swapi, се създават "временни" променливи x и y, в които се запомнят конвертираните стойности на инициализаторите. Размяната се извършва, но само в стековата рамка на swapi.

При предаване на параметрите по псевдоним или по указател, фактическите параметри са променливи или адреси на променливи, за разлика от предаването на параметри по стойност, когато фактическите параметри могат да са изрази в общия случай.

Възможно е някои параметри да се подават по стойност, други по псевдоним или по указател, а също функцията да връща резултат и чрез оператора return. Примери ще бъдат дадени в следващите части на изложението. Ще бъдат обсъдени също предимствата и недостатъците на всеки от начините за предаване на параметрите.

Ако функция не връща резултат чрез return (типът й е void), се нарича още **процедура**.

Разгледаните програми се състояха от две функции. По-сериозните приложения съдържат повече функции. Подредбата им може да започва с main, след която в произволен ред да се дефинират останалите функции. В този случай, дефиницията на main трябва да се предшества от *декларациите* на останалите функции. Декларацията на една функция се състои от заглавието й, следвано от ;. Имената на формалните параметри могат да се пропуснат. Например, програмата от Zad69_1.cpp може да се запише във вида:

```
// Program Zad69_1.cpp
#include <iostream.h>
#include <iomanip.h>
void swapi(double&, double&); // декларация на swapi
int main()
{cout << "a, b, c, d= ";
 double a, b, c, d;
 cin >> a >> b >> c >> d;
```

```
 cout << setprecision(2) << setiosflags(ios :: fixed);
 cout << setw(10) << a << setw(10) << b
      << setw(10) << c << setw(10) << d << '\n';
 swapi(a, b);
 swapi(c, d);
 cout << setw(10) << a << setw(10) << b
      << setw(10) << c << setw(10) << d << '\n';
 return 0;
}
void swapi(double& x, double& y) // дефиниция на swapi
{double work = x;
 x = y;
 y = work;
 return;
}
```

8.3 Дефиниране на функции

Синтаксис

Дефиницията на функция се състои от две части: заглавие (прототип) и тяло. Синтаксисът й е показан на Фиг. 8.4.

Дефиниране на функция

[<модификатор>]$_{опц}$[<тип_на_функция>] $_{опц}$ <име_на_функция>
(<формални_параметри>)
{<тяло>
}

където

<модификатор>::= inline|static| ...
<тип_на_функцията> ::= <име_на_тип> | <дефиниция_на_тип>
<име_на_функция> ::= <идентификатор>
<формални_параметри> :: <празно> | void |
<параметър> {, <параметър>}
<параметър> ::= <тип>[& |$_{opc}$ * [const]$_{opc}$] $_{opc}$ <име_на_параметър>
<тип> ::= <име_на_тип>

```
<име_на_параметър> ::= <идентификатор>
<тяло> ::= <редица_от_оператори_и_дефиниции>
```

Фиг. 8.4 Дефиниция на функция

Модификаторите са спецификатори, които задават препоръка за компилатора (inline), класа памет (extern или static) и др. характеристики. Ще дадем примери в следващите разглеждания. Ако <модификатор> е пропуснат, подразбира се extern.

Типът на функцията е произволен тип без масив и функционален, но се допуска да е указател към такива обекти (в широкия смисъл на думата). Ако е пропуснат, подразбира се int.

Името на функцията е произволен идентификатор. Допуска се неоднозначност.

Списъкът от формални параметри (нарича се още **сигнатура**) може да е празен или void. Например, следната функция извежда текст:

```
void printtext(void)
{cout << “C++ Programming Language \n”
 cout << “B. Stroustrup \n”;
 return;
}
```

В случай, че списъкът е непразен, имената на параметрите трябва да са различни. Те заедно с името определят еднозначно функцията. Формалните параметри са: *параметри – стойности*, *параметри – указатели* и *параметри – псевдоними*. Името на параметъра се предшества от тип.

Примери:

```
int a, int const& b, double& x, int const * y, const int* a
```

Засега няма да използваме параметри, специфицирани със const.

Тялото на функцията е редица от дефиниции и оператори. Тя описва алгоритъма, реализиращ функцията. Може да съдържа един или повече оператора return.

Операторът return (Фиг. 8.5) връща резултата на функцията в мястото на извикването.

Оператор return

Синтаксис

return [<израз>]$_{опц}$

където

- return е запазена дума;

- <израз> е произволен израз от тип <тип_на_функцията> или съвместим с него. Ако типът на функцията е void, <израз> се пропуска. В този случай е възможно и return да се пропусне.

Семантика

Пресмята се стойността на <израз>, конвертира се до типа на функцията (ако е възможно) и връщайки получената стойност в мястото на извикването на функцията, прекратява изпълнението ѝ.

Фиг. 8.5 Оператор return

Забележка: Ако функцията не е от тип void, тя задължително трябва да върне стойност. Това означава, че операторът return трябва да се намира във всички разклонения на тялото. В противен случай, повечето компилатори ще изведат съобщение или предупреждение за грешка. Възможно е обаче функцията да върне случайна стойност, което е лошо. По-добре е функцията да върне някаква безобидна стойност, отколкото случайна.

Функциите могат да се дефинират в произволно място на програмата, но не и в други функции. Преди да се извика една функция, тя трябва да е "позната" на компилатора. Това става, като дефиницията на функцията се постави пред main или когато функцията се дефинира на произволно място в частта за дефиниране на функции, а преди дефинициите на функциите се постави само нейната декларация (Фиг. 8.6).

Декларация на функция

<декларация_на_функция> ::=
[<модификатор>][<тип_на_резултата>]<име_на_функция>
([<формални_параметри>]);

Фиг. 8.6 Декларация на функция

Възможно е имената на параметрите във <формални_параметри> да се пропуснат.

Семантика

Описанието на функция задава параметрите, които носят входа и изхода, типа на резултата, а също и алгоритъма, за реализиране на действията, което функцията дефинира. Параметрите-стойности най-често задават входа на функцията. Параметрите-указатели и псевдоними са входно-изходните параметри за нея. Алгоритъмът се описва в тялото на функцията. Изпълнението на функцията завършва при достигане на края на тялото или след изпълнение на оператор return [<израз>]$_{опц}$;.

8.4 Обръщение към функция

Синтаксис

<обръщение_към_функция> ::=
<име_на_функция>()|
<име_на_функция>(void)|
<име_на_функция>(<фактически_параметри>)

където <фактически_параметри> са толкова на брой, колкото са формалните параметри. Освен по брой, формалните и фактическите параметри трябва да си съответстват по тип, по вид и по смисъл.

Съответствието по тип означава, че типът на i-тия фактически параметър трябва да съвпада (да е съвместим) с типа на i-тия формален параметър. Съответствието по вид се състои в следното: ако формалният параметър е параметър-указател, съответният му фактически параметър задължително е променлива или адрес на променлива, ако е параметър-псевдоним, съответният му фактически параметър задължително е променлива (за реализацията Visual C++, 6.0 от същия тип) и ако е параметър-стойност – съответният му фактически параметър е израз.

Семантика

Обръщението към функция е унарна операция с най-висок приоритет и с операнд - името на функцията. Последното пък е указател със

стойност адреса на мястото в паметта където е записан програмният код на функцията. Ако функцията определя процедура, обръщението към нея се оформя като оператор (завършва с ;). Опитът за използването ѝ като израз предизвиква грешка. Ако функцията връща резултат както чрез return, така и чрез някой от формалните си параметри, обръщението към нея може да се разглежда и като оператор, и като израз. И ако функцията връща резултат единствено чрез оператора return, обръщението към нея има единствено смисъла на израз. Използването му като оператор не води до грешка, но не предизвиква видим резултат.

Обръщението към функция предизвиква генериране на нова стекова рамка и се осъществява на следните два етапа:

1. Свързване на формалните с фактическите параметри

За целта първият формален параметър се свързва с първия фактически, вторият формален параметър се свързва с втория фактически и т.н. последният формален параметър се свързва с последния фактически параметър. Свързването се реализира по различни начини в зависимост от вида на формалния параметър.

а) формален параметър – стойност

В този случай се намира стойността на съответния му фактически параметър. В стековата рамка на функцията за формалния параметър се отделя толкова памет, колкото типът му изисква и в нея се откопирва стойността на фактическия параметър.

б) формален параметър – указател

В този случай в стековата рамка на функцията за формалния параметър се отделят 4В, в които се записва стойността на фактическия параметър, която е адрес на променлива. Действията, описани в тялото се изпълняват със съдържанието на формалния параметър - указател. По такъв начин е възможна промяна на стойността на променливата, чийто адреа е предаден като фактически параметър.

в) формален параметър – псевдоним

Формалният параметър-псевдоним се свързва с адреса на фактическия. За него в стековата рамка на функцията памет не се отделя. Той просто “прелита” и се “закачва” за фактическия си параметър. Действията с него се извършват над фактическия параметър.

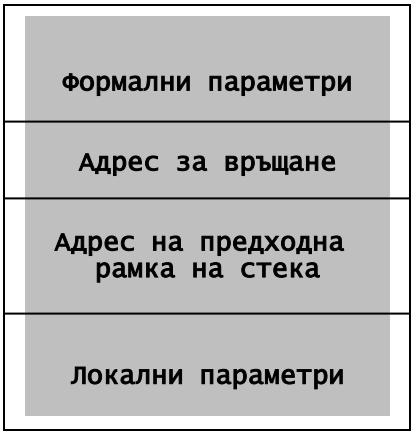
2. Изпълнение на тялото на функцията

Аналогично е на изпълнението на блок.

При всяко обръщение към функция в програмния стек се включва нов "блок" от данни. В него се съхраняват формалните параметри на функцията, нейните локални променливи, а също и някои "вътрешни" данни като return-адреса и др. Този блок се нарича **стекова рамка на функцията**.

В дъното на стека е стековата рамка на main. На върха на стека е стековата рамка на функцията, която се обработва в момента. Под нея е стековата рамка на функцията, извикала функцията, обработваща се в момента. Ако изпълнението на една функция завършва, нейната стекова рамка се отстранява от стека.

Видът на стековата рамка зависи от реализацията. С точност до наредба, тя има вида:



8.7 Стекова рамка

Област на идентификаторите в програмата на C++

Идентификаторите означават имена на константи, променливи, формални параметри, функции, класове. Най-общо казано, има три вида области на идентификаторите: *глобална*, *локална* и *област за клас*. Областите се задават *неявно* – чрез позицията на идентификатора в програмата и *явно*

– чрез декларация. Отново разглеждането ще е непълно, заради пропускането на класовете и явното задаване на област.

Глобални идентификатори

Дефинираните пред всички функции константи и променливи могат да се използват във всички функции на модула, освен ако не е дефиниран локален идентификатор със същото име в някоя функция на модула. Наричат се **глобални идентификатори**, а областта им – **глобална.**

Локални идентификатори

Повечето константи и променливи имат локална област. Те са дефинирани вътре във функциите и не са достъпни за кода в другите функции на модула. Областта им се определя според общото правило – започва от мястото на дефинирането и завършва в края на оператора (блока), в който идентификаторът е дефиниран. Формалните параметри на функциите също имат локална видимост. Областта им е тялото на функцията.

В различните области могат да се използват еднакви идентификатори. Ако областта на един идентификатор се съдържа в областта на друг, последният се нарича нелокален за първоначалния. В този случай е в сила правилото: Локалният идентификатор “скрива” нелокалния в областта си.

Областта на функция започва от нейното дефиниране и продължава до края на модула, в който функцията е дефинирана. Ако дефинициите на функциите са предшествани от тяхните декларации, редът на дефиниране на функциите в модула не е от значение – функциите са видими в целия модул. Препоръчва се също дефинирането на заглавен файл с прототипите (декларациите) на използваните функции.

8.5 Масивите като формални параметри

Едномерни масиви

Съществуват различни начини за задаване на формални параметри от тип едномерен масив.

а) традиционен

Дефиницията

```
T a[]
```

където T е скаларен тип, задава параметър a от тип едномерен масив с базов тип T. Може да се укаже горна граница на масива, но компилаторът я пренебрегва.

Примери:

```
int a[]     - a е параметър от тип масив от цели числа,
int a[10]   - еквивалентна е на int a[],
double b[]  - b е параметър от тип масив от реални числа,
char c[]    - c е параметър от тип масив от символи.
```

б) чрез указател

Дефиницията

```
T* p
```

където T е скаларен тип, задава параметър p от тип указател към тип T. От връзката между масив и указател следва, че тази дефиниция може да се използва и за дефиниране на формален параметър от тип масив.

Примери: Следните дефиниции на формални параметри са еквивалентни на тези от примера по-горе:

```
int* a      - a е параметър от тип указател към int
double* b   - b е параметър от тип указател към double.
char* c     - c е параметър от тип указател към char.
```

И в двата случая фактическият параметър се указва с името на едномерен масив от същия тип. Необходимо е също на функцията да се подаде като параметър и размерът на масива.

Задача 70. Да се напишат функции, които въвеждат и извеждат елементите на едномерен масив от цели числа. Като се използват тези функции да се напише програма, която въвежда редица от естествени числа, след което я извежда, а също извежда най-големия общ делител на елементите на редицата.

Програма Zad70.cpp решава задачата.

```
// Program Zad70.cpp
#include <iostream.h>
int gcd(int, int);
void readarr(int, int[]);
void writearr(int, int[]);
int main()
```

```
{cout << "n= ";
 int n;
 cin >> n;
 int a[20];
 readarr(n, a);
 writearr(n, a);
 int x = a[0];
 for (int i = 1; i <= n-1; i++)
  x = gcd(x, a[i]);
 cout << "gcd = " << x << '\n';
 return 0;
}
int gcd(int a, int b)
{while (a != b)
 if (a > b) a = a-b; else b = b-a;
 return a;
}
void readarr(int m, int arr[])
// m е размерността на масива
// arr е едномерен масив
{for (int i = 0; i <= m-1; i++)
 {cout << "arr[" << i << "]= ";
  cin >> arr[i];
 }
}
void writearr(int m, int arr[])
// m е размерността на масива
// arr е едномерен масив
{for (int i = 0; i <= m-1; i++)
 cout << "arr[" << i << "]= " << arr[i] << '\n';
}
```

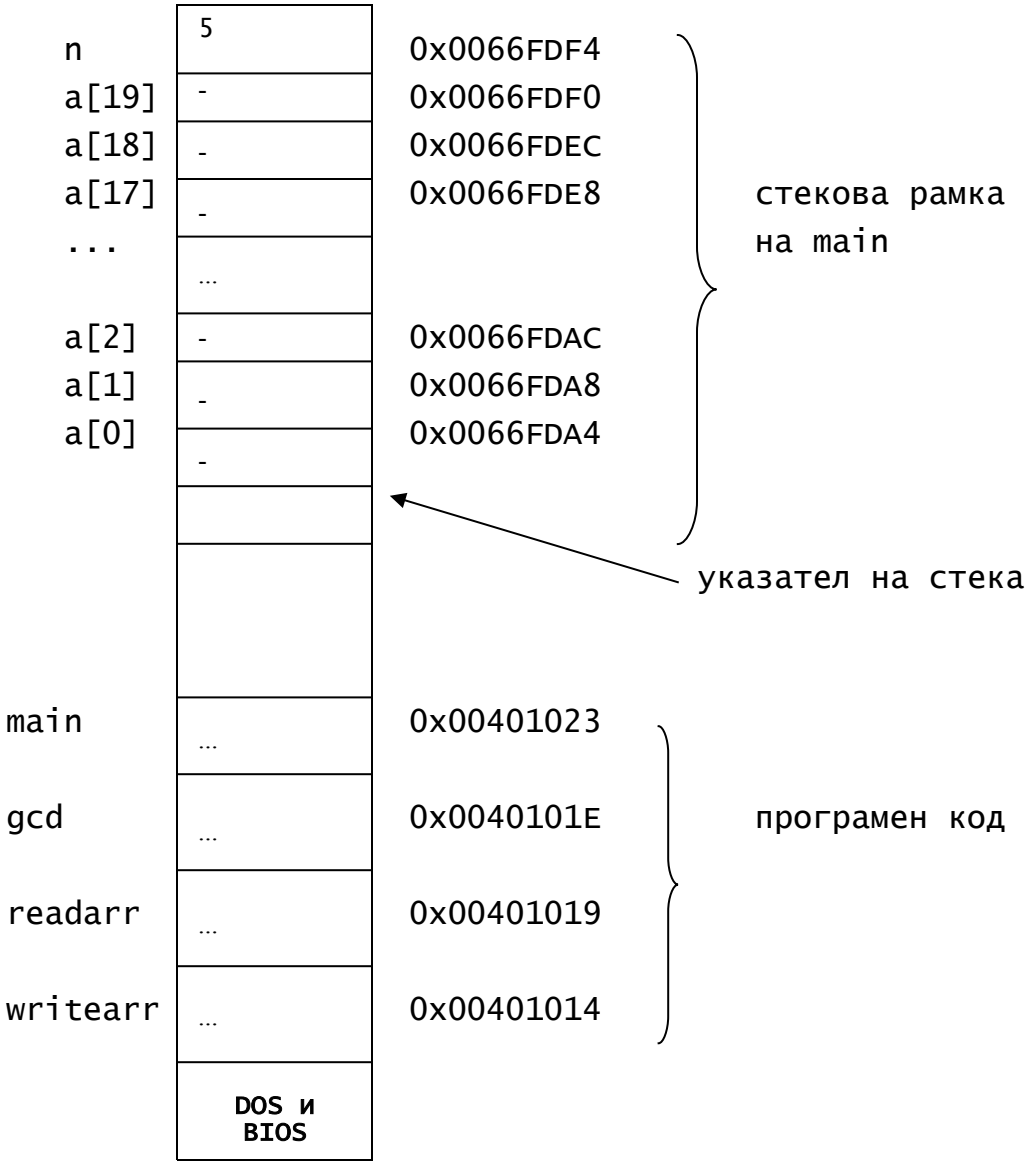
Изпълнение на програмата

Фрагментът

```
cout << "n= ";
int n;
cin >> n;
```

```
int a[20];
```

дефинира и въвежда стойност на n, а също дефинира променлива a от тип масив. Нека за n е въведено 5. В резултат е създадена стековата рамка на main. ОП до този момент има вида:



Обръщението readarr(n, a); се реализира като се свързват формалните с фактическите параметри и се изпълни тялото. За целта се формира нова стекова рамка – тази на readarr, в която за формалния параметър arr се отделят 4B, в която памет се откопирва стойността на фактическия параметър a (адресът на a[0]), за m се отделят също 4B, в които се откопирва 5 – стойността на фактическия параметър n. Тялото на функцията се изпълнява като блок. Операторът за цикъл

```
     for (int i = 0; i <= m-1; i++)
     {cout << "arr[" << i << "]= ";
      cin >> arr[i];
      }
```

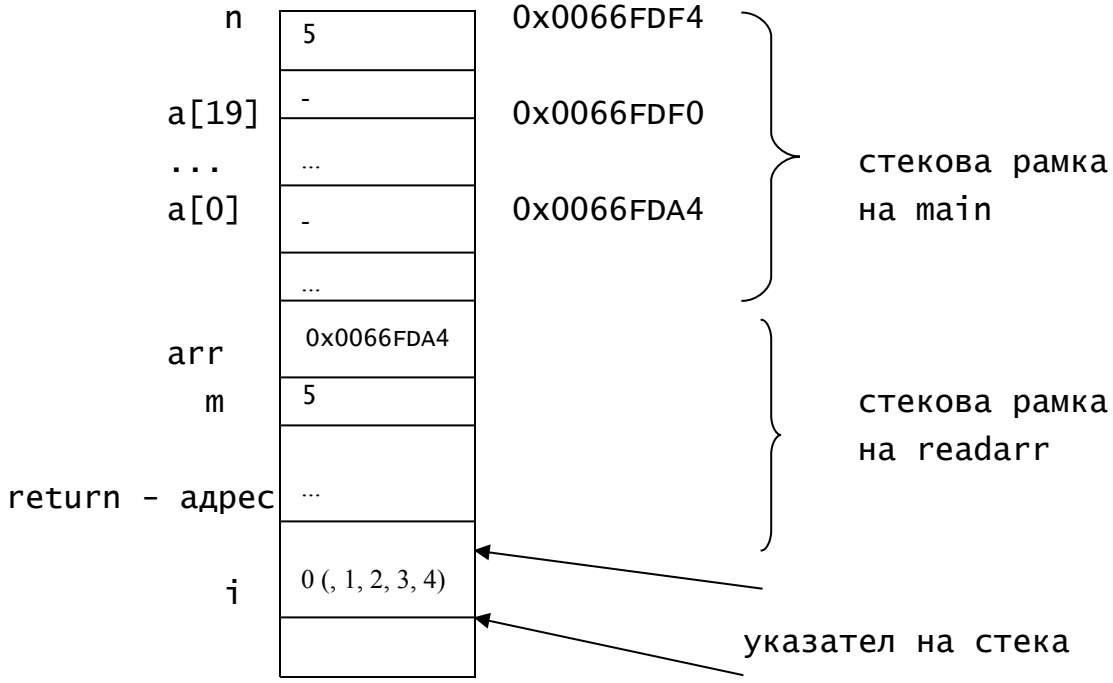
е еквивалентен на

```
     for (int i = 0; i <= m-1; i++)
     {cout << "arr[" << i << "]= ";
      cin >> *(arr + i);
      }
```

и се изпълнява по следния начин: За цялата променлива i се отделят 4в в стековата рамка на readarr. i последователно приема стойностите 0, 1, ..., 4 и за всяка стойност се изпълнява блокът

```
     {cout << "arr[" << i << "]= ";
      cin >> *(arr + i);
      }
```

Операторът cin >> *(arr + i); въвежда стойност на индексираната променлива a[i], тъй като arr + i е адреса на i-тия елемент на a, а *(arr+i) е неговата стойност. Така във функцията се работи с формалния параметър arr, а в действителност действията се изпълняват с фактическия параметър – едномерния масив a. Функцията readarr работи с масива a, а не с негово копие.



След достигане на края на функцията изпълнението ѝ завършва и се освобождава стековата ѝ рамка. В резултат, първите 5 елемента на масива a получават текущи стойности. Операторът

```
writearr(n, a);
```

се изпълнява по аналогичен начин. Отново се работи с фактическия параметър - масива a, а не с негово копие. В този случай, елементите a[0], a[1], ..., a[n-1] на a само се сканират и извеждат. Не се извършват промени над тях. За да ги защитим от неправомерен достъп, е добре формалният параметър arr да дефинираме като указател към цяла константа, т.е. като const int arr[]. Тогава всеки опит да се променя arr[i] (i = 0, 1, ..., n-1) в writearr ще предизвика грешка.

Фрагментът

```
int x = a[0];
for (int i = 1; i <= n-1; i++)
   x = gcd(x, a[i]);
```

намира най-големия общ делител на елементите на редицата.

В заглавията на последните две процедури горните граници на индексите могат явно да се укажат. Например

```
void writearr(int m, int arr[20])
```

и

```
void readarr(int m, int arr[20])
```

са валидни заглавия, но компилаторът не се нуждае от горната граница. Трябват му само скобите [], за да разпознае параметър от тип масив. Може също да се използва второто представяне на формален параметър от тип масив, т.е.

```
void writearr(int m, int* arr)
```

и

```
void readarr(int m, int* arr)
```

Тези представяния на формалните параметри са напълно еквивлентни.

Забележки:

1. Функциите readarr и writearr работят с направо с масива a, а не с негови копия. Промените на елементите на масива се запазват след излизане от функцията.
2. Размерът на масивът не може да се разбере от неговото описание. Затова се налага използването на допълнителния параметър m в списъка от аргументи на функциите. Последното не

се отнася за масивите, представляващи символни низове, тъй като те завършват със знака за край на низ ‘\0’.

Задача 71. Да се напише функция len(char* s), която намира дължината на символен низ, а също функция eqstrs(char*, char*), която сравнява два символни низа за лексикографско равно.

Функциите Zad71_1 и Zad71_2 решават задачата.

```
// Function Zad71_1
int len(char* s)
{int k = 0;
 while(*s)
 {k++;
  s++;
 }
 return k;
}

// Function Zad71_2
bool eqstrs(char* str1, char* str2)
{while (*str1 == *str2 && * str1)
  {str1++; str2++;}
 return *str1 == *str2;
 }
```

Обърнете внимание, че тъй като всеки низ завършва със символа ‘\0’, който се интерпретира като false, изразът *s в оператора за цикъл while на функцията len, ще бъде истина и тялото ще се изпълнява до достигане на края на низа. Аналогична конструкция имаме и при дефиницията на функцията eqstrs.

Задача 72. Да се напише булева функция, която проверява дали цялото число x е елемент на редицата от цели числа a_0, a_1, ..., a_{n-1}.

Функцията Zad72 решава задачата.

```
// Function Zad72
```

```
bool search(int n, int a[], int x)
{int i = 0;
 while (a[i] != x && i < n-1)i++;
 return a[i]==x;
}
```

Допълнение: Обръщението

search(m, b, y)

проверява дали елементът y се съдържа в редицата $b_0, b_1, \ldots, b_{m-1}$, а

search(k, b + m, y)

проверява дали y се съдържа в подредицата $b_m, b_{m+1}, \ldots, b_{m+k-1}$.

Еквивалентна дефиниция на тази функция е:

```
bool search(int n, int* a, int x)
{int i = 0;
 while (*(a+i) != x && i < n-1)i++;
 return *(a+i)==x;
 }
```

Задача 73. Да се напише функция, която проверява дали редицата от цели числа $a_0, a_1, \ldots, a_{n-1}$ е монотонно намаляваща.

Функцията Zad73 решава задачата.

```
// Function Zad73
bool monnam(int n, int a[])
{int i = 0;
 while (a[i] >= a[i+1] && i < n-2)i++;
 return a[i] >= a[i+1];
}
```

или

```
bool monnam(int n, int* a)
{int i = 0;
 while (*(a+i) >= *(a+i+1) && i < n-2)i++;
 return *(a+i) >= *(a+i+1);
}
```

Допълнение: Обръщението

monnam(m, b)

проберява дали редицата b_0, b_1, ..., b_{m-1} е монотонно намаляваща, а
  monnam(k, b + m)
- дали подредицата b_m, b_{m+1}, ..., b_{m+k-1} на b_0, b_1, ..., b_{m-1} е монотонно намаляваща.

Задача 74. Да се напише функция, която проверява дали редицата от цели числа a_0, a_1, ..., a_{n-1} се състои от различни елементи.

Функцията Zad74 решава задачата.

```
// Function Zad74
bool differ(int n, int a[]
)
{int i = -1;
 bool b; int j;
 do
 {i++; j = i;
  do
  {j++;
   b = a[i] != a[j];
  }while (b && j < n-1);
  }while (b && i < n-2);
  return b;
 }
```

Допълнение: Обръщението
  differ(m, b, y)
проберява дали редицата b_0, b_1, ..., b_{m-1} се състои от различни елементи, а
  differ(k, b + m)
- дали подредицата b_m, b_{m+1}, ..., b_{m+k-1} на b_0, b_1, ..., b_{m-1} се състои от различни елементи.

Задача 75. Да се напише програма, която въвежда две числови редици, сортира ги във възходящ ред, слива ги и извежда получената редица.

Програма Zad75.cpp решава задачата. За целта са дефинирани следните функции:

readarr – въвежда числова редица
writearr – извежда числова редица върху екрана
sortarr - сортира във възходящ ред елементите на числова редица
mergearrs - слива числови редици.

```
// Program Zad75.cpp
#include <iostream.h>
#include <iomanip.h>
void writearr(int, double[]);
void readarr(int, double[]);
void sortarr(int, double[]);
void mergearrs(int, double[], int, double[], int&, double[]);

int main()
{cout << "n= ";
 int n;
 cin >> n;
 double a[20];
 readarr(n, a);
 cout << endl;
 writearr(n, a);
 cout << endl;
 sortarr(n, a);
 cout << endl;
 writearr(n, a);
 cout << "m= ";
 int m;
 cin >> m;
 double b[30];
 readarr(m, b);
 cout << endl;
 writearr(m, b);
 cout << endl;
 sortarr(m, b);
```

```
 cout << endl;
 writearr(m, b);
 cout << endl;
 int p;
 double c[50];
 mergearrs(n, a, m, b, p, c);
 writearr(p, c);
 return 0;
}
void writearr(int m, double arr[])
{cout << setprecision(3) << setiosflags(ios::fixed);
 for (int i = 0; i <= m-1; i++)
   cout << setw(10) << arr[i];
 cout << "\n";
}
 void readarr(int m, double arr[])
 {for (int i = 0; i <= m-1; i++)
 {cout << "arr[" << i << "]= ";
  cin >> arr[i];
 }
 }
  void sortarr(int n, double a[])
 {for (int i = 0; i <= n-2; i++)
 {int k = i;
  double min = a[i];
  for (int j = i+1; j <= n-1; j++)
    if (a[j] < min)
    {min = a[j];
     k = j;
    }
  double x = a[i]; a[i] = a[k]; a[k] = x;
 }
}
void mergearrs(int n, double a[], int m, double b[],
               int& k, double c[])
{int i = 0, j = 0;
```

```
    k = -1;
    while (i <= n-1 && j <= m-1)
    if (a[i] <= b[j])
    {k++;
     c[k] = a[i];
     i++;
    }
    else
    {k++;
     c[k] = b[j];
     j++;
    }
    int l;
    if (i > n-1)
      for (l = j; l <= m-1; l++)
      {k++;
       c[k] = b[l];
      }
    else
      for (l = i; l <= n-1; l++)
      {k++;
       c[k] = a[l];
      }
      k++;
   }
```

Многомерни масиви

Когато многомерен масив трябва да е формален параметър на функция, в описанието му трябва да присъстват като константи всички размери с изключение на първият. Например, декларацията

```
void readarr2(int n, int matr[][20]);
```

определя matr като двумерен масив (редица от двадесеторки от цели числа). Описанието

```
int (*matr)[20]
```

е еквивалентно на

```
int matr[][20]
```

Скобите, ограждащи *matr, са задължителни. В противен случай, тъй като [] е с по-висок приоритет от *, int *matr[20] ще се интерпретира като “matr е масив с 20 елемента от тип *int”.

Задача 76. Да се напише програма, която въвежда квадратна матрица от цели числа, след което я извежда като увеличава всеки от елементите на матрицата над главния диагонал с 5 и намалява всеки от елементите под главния диагонал с 5.

Програма Zad76.cpp решава задачата. Тя дефинира функциите:
readarr2 – въвежда квадратна матрица
writearr2 – извежда квадратна матрица
transff - увеличава всеки от елементите на матрицата над главния диагонал с 5 и намалява всеки от елементите под главния диагонал с 5.

```
// Program Zad76.cpp
#include <iostream.h>
#include <iomanip.h>
void readarr2(int, int[][10]);
void writearr2(int, int[][10]);
void transff(int, int[][10]);
int main()
{int a[10][10];
 cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1 || n > 10)
 {cout << "Incorrect input! \n";
  return 1;
 }
 readarr2(n, a);
 cout << '\n';
```

```
 writearr2(n, a);
 cout << '\n';
 transff(n, a);
 writearr2(n, a);
 return 0;
}
void readarr2(int n, int arr[][10])
{for (int i = 0; i <= n-1; i++)
  for (int j = 0; j <= n-1; j++)
   cin >> arr[i][j];
}
void writearr2(int n, int arr[][10])
{for (int i = 0; i <= n-1; i++)
 {for (int j = 0; j <= n-1; j++)
   cout << setw(5) << arr[i][j];
  cout << "\n";
 }
}
void transff(int n, int arr[][10])
{int i, j;
 for (i = 1; i <= n-1; i++)
  for (j = 0; j <= i-1; j++)
     arr[i][j] = arr[i][j] - 5;
 for(i = 0; i <= n-2; i++)
  for(j = i+1; j <= n-1; j++)
     arr[i][j] = arr[i][j] + 5;
}
```

Обръщението

```
transff(k, a + m);
```

ще извърши същото действие над квадратната подматрица на дадената матрица:

$$\begin{pmatrix} a_{m,0} & a_{m,1} & \dots & a_{m,k-1} \\ a_{m+1,0} & a_{m+1,1} & \dots & a_{m+1,k-1} \\ \dots & \dots & \dots & \dots \\ a_{m+k-1,0} & a_{m+k-1,1} & \dots & a_{m+k-1,k-1} \end{pmatrix}$$

Задача 77. Да се напише програма, която въвежда редица от думи не по-дълги от 14 знака и дума, също не по-дълга от 14 знака. Програмата да проверява дали думата се среща в редицата. За целта да се оформят подходящи функции.

Програма Zad77.cpp решава задачата. В нея са дефинирани функциите:
void readarrstr(int n, char s[][15]) - въвежда редица от n думи,
bool search(int n, char s[][15], char* x) - търси думата x в редицата s от n думи. За целта използва помощната функция
bool eqstrs(char* str1, char* str2);
от Задача 71.

```
// Program Zad77.cpp
#include <iostream.h>
#include <string.h>
void readarrstr(int, char [][15]);
bool eqstrs(char*, char*);
bool search(int, char [][15], char*);
int main()
{char a[20][15];
 cout << "n= ";
 int n;
 cin >> n;
 readarrstr(n, a);
 cout << "word: ";
 char word[15];
 cin >> word;
 if (search(n, a, word)) cout << "yes \n";
 else cout << "no \n";
 return 0;
}
void readarrstr(int n, char s[][15])
{for(int i = 0; i <= n-1; i++)
 {cout << "s[" << i << "]= ";
  cin >> s[i];
```

```
 }
}
bool eqstrs(char* str1, char* str2)
 {while (*str1 && *str1 == *str2)
   {str1++;
    str2++;
   }
  if(*str1 != *str2) return false;
  else return true;
 }
bool search(int n, char s[][15], char* x)
{int i = 0;
 while (!eqstrs(s[i], x) && i < n-1) i++;
 return eqstrs(s[i], x);
}
```

Задача 78. Да се напише програма, която умножава две матрици.

Програма Zad78.cpp решава задачата. Тя дефинира следните функции:
readarr2 – въвежда матрица,
writearr2 – извежда матрица,
multmatr – умножава матрици.

```
// Program Zad78.cpp
#include <iostream.h>
#include <iomanip.h>
void readarr2(int n, int m, double [][30]);
void writearr2(int n, int m, double [][30]);
void multmatr(int, int, int, double [][30],
           double [][30], double [][30]);

int main()
{double a[10][30], b[20][30], c[10][30];
 cout << "n= ";
 int n;
 cin >> n;
```

```
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
}
if (n < 1 || n > 10)
{cout << "Incorrect input! \n";
 return 1;
}
cout << "m= ";
int m;
cin >> m;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
}
if (m < 1 || m > 30)
{cout << "Incorrect input! \n";
 return 1;
}
 cout << "k= ";
int k;
cin >> k;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
}
if (k < 1 || k > 30)
{cout << "Incorrect input! \n";
 return 1;
}
readarr2(n, m, a);
writearr2(n, m, a);
cout << "\n";
readarr2(m, k, b);
cout << "\n";
writearr2(m, k, b);
```

```
 cout << "\n";
 multmatr(n, m, k, a, b, c);
 writearr2(n, k, c);
 return 0;
}
void readarr2(int n, int m, double arr [][30])
{for (int i = 0; i <= n-1; i++)
   for (int j = 0; j <= m-1; j++)
      cin >> arr[i][j];
 return;
}
void writearr2(int n, int m, double arr[][30])
{cout << setprecision(3) << setiosflags(ios::fixed);
 for (int i = 0; i <= n-1; i++)
 {for (int j = 0; j <= m-1; j++)
   cout << setw(10) << arr[i][j];
  cout << "\n";
 }
 return;
}
void multmatr(int n, int m, int k, double a[][30],
              double b[][30], double c[][30])
{for (int i = 0; i <= n-1; i++)
  for (int j = 0; j <= m-1; j++)
  {c[i][j] = 0;
   for (int p = 0; p <= m-1; p++)
      c[i][j] += a[i][p] * b[p][j];
  }
}
```

8.6 Масивите като върнати оценки

Въпреки, че масивите могат да са параметри на функции, функциите **не могат** да са от тип масив. Възможно е обаче да са от тип указател. Това позвлява дефинирането на функции, които връщат масиви.

Пример: В следващата програма е дефинирана функцията readarr, която въвежда стойности на едномерен масив. Тя връща резултат не само чрез променливата от тип масив arr, но и чрез оператора return. Това позволява обръщенията към нея да служат както за оператори, така и за изрази.

```
#include <iostream.h>
void writearr(int, int[]);
int* readarr(int, int[]);
int main()
{cout << "n= ";
 int n;
 cin >> n;
 int a[20];
 int* p = readarr(n, a);
 writearr(n, p);
 cout << endl;
 return 0;
}
void writearr(int m, int arr[])
{for (int i = 0; i <= m-1; i++)
 cout << "arr[" << i << "]= " << arr[i] << '\n';
return;
}
int* readarr(int m, int arr[])
{for (int i = 0; i <= m-1; i++)
 {cout << "arr[" << i << "]= ";
  cin >> arr[i];
 }
return arr;
}
```

Въпрос: Допустима ли е конструкцията: readarr(n, a)[i], където i е цяло число от 0 до n-1? Ако това е така, какъв е резултатът от изпълнението му?

Задача 79. Да се напише функция, която намира и връща като резултат конкатенацията на два низа. Функцията да променя първия си

аргумент като в резултат той също да съдържа конкатенацията на низовете.

Програма Zad79.cpp решава задачата.

```
// Program Zad79.cpp
#include <iostream.h>
int len(char*);
char *cat(char*, char*);
int main()
{char s1[100];
 cout << "s1= ";
 cin >> s1;
 cout << "s2= ";
 char s2[100];
 cin >> s2;
 cout << cat(s1, s2) << "  " << s1 << '\n';
 return 0;
}
int len(char* s)
{int k = 0;
 while (*s)
 {k++; s++;
 }
 return k;
}
char* cat(char *s1, char *s2)
{int i = len(s1);
 while (*s2)
 {s1[i] = *s2;
  i++;
  s2++;
}
 s1[i] = '\0';
 return s1;
}
```

Задачи

Задача 1. Въпреки многото й недостатъци, следващата програма е доста поучителна. Тя дефинирана функцията readarr, която има за формален параметър броя на елементите на масива и връща едномерен масив, определен чрез указател към първия му елемент.

```
#include <iostream.h>
int a[20];
void writearr(int, int[]);
int* readarr(int);
int main()
{cout << "n= ";
 int n;
 cin >> n;
 int* p = readarr(n);
 writearr(n, p);
 cout << '\n';
 return 0;
}
void writearr(int m, int arr[])
{for (int i = 0; i <= m-1; i++)
 cout << "arr[" << i << "]= " << arr[i] << '\n';
}
int* readarr(int m)
{for (int i = 0; i <= m-1; i++)
{cout << "a[" << i << "]= ";
 cin >> a[i];
}
return a;
}
```

Извършете експерименти с тази програма.

Задача 2. Да се напише програма, която въвежда полиномите:

$$P_n(x) = a_o x^{n-1} + a_1 x^{n-2} + \ldots + a_{n-2} x^1 + a_{n-1},$$

$$R_k(x) = r_o x^{k-1} + r_1 x^{k-2} + \ldots + r_{k-2} x^1 + r_{k-1}$$

$$Q_n(x) = b_o x^{n-1} + b_1 x^{n-2} + \ldots + b_{n-2}x^1 + b_{n-1},$$

и реалната променлива x и намира и извежда стойностите на полиномите в x.

Задача 3. Да се напише програма, която въвежда стойности на редиците:

$a_0, a_1, \ldots, a_{n-1},$

$b_0, b_1, \ldots, b_{m-1},$

$c_0, c_1, \ldots, c_{p-1}$

и намира и извежда AR1, AR2, BR1, BR2, CR1 и CR2, където за дадена редица $x_0, x_1, \ldots, x_{k-1}$

$$XR1 = \frac{1}{k}\sum_{i=0}^{k-1} x_i, \qquad XR2 = \frac{1}{k}\sqrt{\sum_{i=0}^{k-1}(x_i - XR1)^2}.$$

Задача 4. Да се напише функция, която намира разстоянието между две точки в равнината, зададени чрез координатите си (x1, y1) и (x2, y2). Като се използва тази функция да се напише програма, която чете координатите на n точки ($n \geq 1$) от равнината и намира и извежда разстоянието между всеки две от тях.

Задача 5. да се напише функция, която връща стойност true, ако a, b и c са страни на триъгълник и false - в противен случай. Като се използва тази функция, да се напише програма, която въвежда стойности на елементите на матрицата A_{3xn} и определя кои от тройките (a[0][i], a[1][i], a[2][i]), i = 0, 1, ..., n-1 могат да служат за страни на триъгълник.

Задача 6. Да се напише функция, която връща стойност true ако редицата от цели числа $x_0, x_1, \ldots, x_{k-1}$ има поне два последователни нулеви елемента. Като се използва тази функция, да се напише програма, която намира и извежда номерата на редовете на матрицата A [n x n], от цели числа, които имат поне два последователни нулеви елемента.

Задача 7. Да се напише функция, която намира сумата на два полинома. Като се използва тази функция, да се напише програма, която намира сумата на всеки два от полиномите:

$$P_n(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x^1 + a_0,$$

$$Q_m(x) = b_{n-1}x^{m-1} + b_{n-2}x^{m-2} + \ldots + b_1 x^1 + b_0,$$

$$R_k(x) = r_{k-1}x^{k-1} + r_{k-2}x^{k-2} + \ldots + r_1 x^1 + r_0,$$

Задача 8. Даден е триъгълник със страни a, b и c. Да се напише програма, която намира медианите на триъгълник, страните на който са медианите на дадения триъгълник.

Упътване: Медианата към страната a на триъгълника е равна на

$$\frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}.$$

Задача 9. Дадени са координатите на върховете на n триъгълника. Да се напише програма, която определя, кой от триъгълниците е с по-голямо лице.

Задача 10. Дадени са естественото число p > 1 и реалните квадратни матрици с размерности n x n - A, B и C. Да се напише програма, която намира матрицата

$$(A \cdot B \cdot C)^p.$$

Допълнителна литература

1. B. Stroustrup, C++ Programming Language. Third Edition, Addison – Wesley, 1997.
2. Ст. Липман, Езикът C++ в примери, "КОЛХИДА ТРЕЙД" КООП, София, 1993.
3. 3. Д. Луис, C/C++ бърз справочник, ИнфоДАР, София, 1998.
4. И. Момчев, К. Чакъров, Програмиране III, C и C++, ТУ, София, 1996.
5. М. Тодорова, Програмиране на Паскал, Полипринт, София, 1993.